VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2016

Martin Vaněk

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Martin Vaněk**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: VRK plus s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Daniel Stříbný**

Konzultant bakalářské práce: Mgr. Robert Kubáček

Datum zadání: 01.09.2015

Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 21. dubna 2016 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 21. dubna 2016

.....

VRK plus s.r.o.
Fr. Lýska 1605/3
700 30 Ostrava-Bělský Les
IČ: 60321580, DIČ: CZ60321580

Chtěl bych poděkovat všem, kteří mi na této práci pomáhali či mi byli inspirací. Největší dík patří hlavně Ing. Danielovi Stříbnému, Mgr. Robertovi Kubáčkovi a všem kolegům z VRK plus s.r.o.

Abstrakt

Toto dílo rozebírá mou bakalářskou práci, která byla vykonána formou odborné praxe ve firmě VRK plus s.r.o. Nejprve krátce popisuje profil firmy, použité technologie pro lepší orientaci v práci a samotný popis vykonané práce. Cílem této praxe bylo vyvinout webovou aplikaci pohodlne.info skrze portletové řešení pro portál Liferay.

Klíčová slova: Liferay; JavaServer Faces; Java EE; Portlet

Abstract

This work portraits my bachelor thesis which was done via proffesional practise in VRK plus s.r.o. company. First, it shortly describes company profile, used technology for better orientation in the thesis and the practise description itself. The goal of this practise was to develop web application pohodlne.info which meant developing portlets for Liferay portal.

Key Words: Liferay; JavaServer Faces; Java EE; Portlet

Obsah

Seznam použitých zkratek a symbolů

EL	–	Expression Language
GUI	–	Graphical User Interface
HTTP	–	Hypertext Transfer Protocol
JSF	–	JavaServer Faces
JSON	–	JavaScript Object Notation
POJO	–	Plain Old Java Object
UUID	–	Universally Unique Identifier
XML	–	Extensible Markup Language
XHTML	–	Extensible Hypertext Markup Language

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Uvnitř této bakalářské práce vás provedu mojí praxí ve firmě VRK plus s.r.o. Ve firmě jsem již nějakou dobu figuroval jako vývojář, proto jsem se rozhodl spojit práci se školou. Moje původní pozice byla obecná a pracoval jsem s různými technologiemi a s různými aplikacemi. Během této praxe jsem měl jasně danou pozici „Java programátor pro Liferay“.

Mým úkolem bylo postupně převést a rozšířit funkcionalitu aktuálně spuštěného systému Smile KLUB. To zahrnovalo nutnou komunikaci a vzájemnou synchronizaci.

Tato práce obsahuje krátkou charakteristiku firmy a seznámení s aplikacemi, které se týkali této praxe. Dále je zde orientační popis použitých technologií a nakonec náplň mé práce.

2 Popis firmy VRK plus s.r.o.

2.1 Historie

Společnost VRK plus s.r.o. byla založena v roce 1993. Zaměřuje se především na tvorbu vlastních informačních systémů a počítačových aplikací určených převážně pro vzdělávací instituce, sportovní kluby či asociace. Pro zákazníky vytváří i webové stránky. Také ke svým, ale i k produktům třetích stran, poskytuje školení.[1]

2.2 pohodlne.info

Informační systém pohodlne.info slouží pro podporu organizací, které se věnují provozování různých zájmových činností. Umožňuje těmto organizacím udržovat kontaktní údaje, platby a další informace o svých členech na jednom místě dostupné online. [1].

Systém pohodlne.info vznikl spojením nově vytvořeného stejnojmenného portálu a existující webové aplikace Smile KLUB. V této práci, pokud nebude řečeno jinak, budu zmiňovat pohodlne.info jako samostatnou část bez KLUBu.

2.2.1 Smile KLUB

Smile KLUB (dále jen KLUB) je webová aplikace, která existovala před vznikem portálu pohodlne.info. KLUB je napsán ve vývojové sadě GWT[1]. Firma plánuje tento systém postupně rušit a převádět jeho funkci na portál pohodlne.info. Jedním z důvodů je zastaralost aplikace a potřeba zavádění nové funkcionality. Dalším důvodem je postupné vymírání projektu GWT a přesun všech vývojářů firmy pod jednotnou platformu – pod portál Liferay.

[1] Google Web Toolkit – http://www.gwtproject.org/

3 Použité technologie

3.1 Liferay Portal

Liferay Portal (dále jen *portál*) je nabízen v komunitní verzi CE zdarma a v EE verzi se servisní podporou a poplatky. My jsme používali CE verzi. Portál shromažďuje, zobrazuje a spravuje tzv. portlety, které reprezentují jednotlivé funkční komponenty webového systému (např. portlet pro správu článků). Portál je snadno rozšiřitelný pomocí široké škály pluginů umožňující vývoj v různých technologiích. Portál podporuje množství aplikačních serverů a databází. Naše vybrané prostředí bylo Apache Tomcat a databáze MySQL.

Skoro veškerá portálová funkcionalita je řešena pomocí vestavěných portletů, které se zobrazují na správných místech (např. správa uživatelů, skupin, stránek, CMS[2]).

3.1.1 Plugin

Plugin rozšiřuje funkci portálu a jsou definovány tyto typy:

- **Portlet** obsahuje konkrétní portlety (pozor neplést *plugin typu portlet* s portletem samotným).
- **Hook** je většinou obsažen i v portlet pluginu a díky něho se dají přepisovat portálové hodnoty, jako jsou třeba lokalizační soubory, nastavení portálu a další věci.
- **Service Builder Portlet** je portlet plugin v kombinace se službou generovanou nástrojem *Service Builder*.
- **Layout template** je rozvržení portletů na stránce.
- **Theme** je grafické téma.
- **Ext** může modifikovat portálové třídy.
- **Web** pro statické stránky.

[2] CMS – content management system

3.2 JSF

JSF je technologie postavená na vzoru *Model View Controller*[3] a je hlavně o komponentách, které jsou reprezentovány, jak na straně klienta (prohlížeč), tak na straně serveru. Komponenty jsou jednoduše znovupoužitelné a umožňují stavbu rozsáhlých webových uživatelských rozhraní. Ke komponentám dále patří rozhraní spravující jejich stav, události, validaci vstupu, navigaci stránek, lokalizaci a přístup k nim.[2]

Webové aplikace v JSF se mohou vyvíjet více způsoby. V tomto projektu jsme v pohledové vrstvě používali tzv. Facelets[4], které prostřednictvím s Expression Language (dále jen *EL*) komunikují s Managed Beany, což jsou klasické třídy, označené pomocí anotace `@ManagedBean` [5], dodržující *JavaBean* konvenci. [3]

3.2.1 Liferay Faces Bridge

Liferay Faces Bridge umožňuje psát portletové aplikace v JSF stejně jako klasické webové, bez větší potřeby znalosti portletových specifik. Projekt Liferay Faces dále také obsahuje pomocné metody, které usnadňují obvyklou práci s portálem.

3.2.2 Primefaces

Primefaces je open-source knihovna obsahující mnohá rozšíření pro původní JSF komponenty. Ty jsou navrženy pro jednoduchou unifikovanou stylizaci, umožňují dobře a jednoduše využívat AJAX[6], podporují HTML5 a snaží se podporovat responzivní design.

3.3 Maven

Maven, plným názvem Apache Maven, je software pro správu projektů. Maven se dá využít od vygenerování projektu, přes správu závislostí, sestavování, až ke spouštění testů a samotnému nasazování. Všechny projekty tvořené během této praxe používaly právě Maven.

Maven je dobře rozšiřitelný pomocí pluginů a Liferay takový plugin poskytuje. V něm je integrována práce s portálem a jeho nástroji (jako je Service Builder).

[3] http://martinfowler.com/eaaCatalog/modelViewController.html
[4] Facelet – deklarační jazyk pro tvoření stromů komponent
[5] http://docs.oracle.com/javaee/7/api/javax/annotation/ManagedBean.html
[6] AJAX – Asynchronous JavaScript and XML

3.4 Service Builder

Je to nástroj od firmy Liferay používaný na generování servisní vrstvy, která je většinou spjata s persistentní vrstvou (nejčastěji databáze). Všechny portálové entity jsou přístupné skrze službu napsanou právě pomocí tohoto nástroje, to značí dobrou integraci s portálem.

Nástroj pomocí deklarativního nastavení v `service.xml` umí vygenerovat model, persistetní a servisní vrstvu. Má také zabudované cachování vygenerovaných entit a umožňuje psát vlastní SQL dotazy. Pro mapování entit do databáze používá tento nástroj framework Hibernate[7].

[7] Hibernate – framework generující tzv. objektově-relační mapování (ORM)

4 Práce na pohodlne.info

Tabulka 1: Vyjádření časové náročnosti

Vykonaná práce	**Počet dnů**
Zaučování potřebných technologií a jejich zkoušení	10
Fáze první: nadstavba nad SMILE Klub	15
Fáze druhá: přesunutí hlavní a nové funkčnosti na portál pohodlne.info	25
Celkem	50

Jak jsem již zmínil, firma nebyla spokojená s dlouhodobou udržitelností aplikace KLUB a chtěla jeho funkčnost postupně spolu se získanými zkušenostmi během provozu a vývoje této aplikace, převádět jako portletové řešení pod portál Liferay. Tento proces byl rozdělen do dvou fází.

4.1 Fáze první: nadstavba nad SMILE Klub

4.1.1 Propojení aplikací

Pro komunikaci mezi aplikacemi jsme využili mechanismu podobnému REST[8] API. Pro spojení s rozhraním KLUBu jsem využil knihovnu *HttpComponents Client*™[9], protože umožňuje jednoduchou práci s HTTP požadavky a odpovědmi. Data byly posílány ve formátu JSON a na parsování jsem si vybral velmi oblíbenou *google-gson*[10], protože umí bez složitého nastavování převést POJO do JSON reprezentace. I když obě tyto aplikace byly psány v Javě a umístěny na stejném serveru, tak v čase psaní této komunikace jsem neměl dostatek znalostí a zkušeností na to, abych tohoto faktoru využil pro lepší a bezpečnější komunikaci uvnitř serveru (nový způsob komunikace je popsán v sekci 4.2.2).

S kolegy jsme přemýšleli nad jednoznačnou identifikací samotných členů mezi systémy. První myšlenka byla použít e-mailovou adresu, ale toto jsme zavrhli, protože uživatel systému pohodlne.info může využít jinou adresu než má uvedenou v evidenci příslušného klubu a nebo ji nemusí mít uvedenou v KLUBu vůbec. Nakonec jsme vymysleli mechanismus tzv. *pi-kód*. V obou systémech se přidal k uživatelům atribut, který při ekvivalenci na obou stranách značí propojení těchto uživatelů. Na mě bylo zajistit unikátnost a implementaci tohoto kódu na straně pohodlne.info.

Kvůli jednoho atributu není rozumné zasahovat do původní funkční portálové struktury či zavádět novou. Naštěstí portál nabízí možnost jednoduše přidat atributy k portálovým entitám

[8] REST – Representational state transfer
[9] http://hc.apache.org/httpcomponents-client-ga/index.html
[10] https://github.com/google/gson

a toho jsem využil. Jediné požadavky na tento kód byly, aby nešel snadno uhodnout a ať je pro každého uživatele unikátní. Vymyslel jsem, že kód bude reprezentovat *hash* vytvořený z kombinace atributů uživatele a pro lepší čitelnost bude zkrácený. Jelikož v tomto případě nejde bezpečnost, ale o generování unikátního identifikátoru, který by zvládl i náhodný generátor, vybral jsem si hashovací funkci SHA-1. Na generaci kódu jsem použil metodu `DigestUtils.sha1Hex` knihovny *Apache Commons Codec*™. Vygenerovaný hash by měl být unikátní, ale jelikož ho zkracuji, pro zajištění unikátnosti zjišťuji jestli tento kód již není někomu přiřazen.

4.1.2 Portlet obsahující hlavní funkčnost

Jakmile mi byly řečeny požadavky a představen návrh GUI, začal jsem s implementací. V té době jsem ještě neměl dostupné funkční rozhraní z KLUBu, proto jsem si vytvořil falešné entity, jejichž strukturu jsem odvodil podle zadaných požadavků.

Cílem bylo udělat 3 záložky – a to členské informace, platby a docházku. Každá záložka obsahovala seznam členů klubu. V řádcích těchto seznamů měly být informace o členech a po výběru konkrétního člena se řádek otevře s detailními informacemi. Po prvním zkoumání, jsem pro tuto funkčnost vybral komponentu `<p:accordionPanel>`[11].

Později mi kolega uvolnil rozhraní na přístup pro data z KLUBu, takže jsem odhadovaná a ukázková data nahradil napojením na KLUB. Pro tuto komunikaci jsem si vytvořil dvě funkce, které umí načíst data z odpovědi na HTTP požadavek a převést tyto data do objektu daného typu.

```
public static <T> T getData(Class<T> type, URI actionURI, Gson gson)
    throws IOException {
  LOG.debug(actionURI.toString());
  try (CloseableHttpClient httpClient = HttpClients.createDefault()) {
  HttpGet httpGet = new HttpGet(actionURI);
  try (CloseableHttpResponse response = httpClient.execute(httpGet)) {
    if (response.getStatusLine().getStatusCode() != HttpStatus.SC_OK) {
      throw new IOException("stranka nenalezena / neni pristup");
    }
      HttpEntity entity = response.getEntity();
      T result = processEntity(entity, type, gson);
      EntityUtils.consume(entity); // vyprazdni entitu
      return result;
    }
  } catch (IOException e) {
    LOG.error("Chyba pri vytvareni httpClienta ,chyba spojeni etc.", e);
    throw e;
  }
```

[11] http://www.primefaces.org/showcase/ui/panel/accordionPanel.xhtml

```
}

private static <T> T processEntity(HttpEntity entity, Class<T> type,
    Gson gson) throws IOException {
  StringBuilder jsonResponse = new StringBuilder();
  try (BufferedReader contentReader = new BufferedReader(new
      InputStreamReader(entity.getContent()))) {
    String line = null;
    while ((line = contentReader.readLine()) != null) {
      jsonResponse.append(line);
    }
    return gson.fromJson(jsonResponse.toString(), type);
  }
```

Výpis 1: Hlavní metody pro komunikaci

Časem se ukázalo, že volba komponenty `<p:accordionPanel>` nebyla nejlepší, protože by se nad ní muselo explicitně vyvinout řazení, filtraci a stránkování. Zjistil jsem, že komponenta `<p:dataTable>` také nabízí funkci rozbalení řádku, ale umí i dobře provádět výše uvedené funkce. Navíc je lépe připravena pro responzivní design.

4.1.3 Nastavení portletu

Funkčnost měla být rozdělena na dvě skupiny – jedna pro trenéra a druhá pro člena klubu. Člen si může prohlížet své údaje, které o něm klub vede, jeho platby, spojení s dalšími členy, skupiny a členství. Tyto údaje může také sledovat u spojení s jinými členy, ve kterých má povolené nahlížení.

Trenér vidí všechny údaje o členech v jeho skupině a ve vybraném klubu, navíc může zadat nové platby, upravovat staré a také může spravovat docházku.

Pohled člena a trenéra obsahuje stejné funkční prvky a jediný rozdíl je nemožnost člena upravovat platby a docházku. Proto jsem se rozhodl stejný portlet využít dvěma způsoby. Na rozhodnutí typu módu jsem využil tzv. *„portlet preference“*, které se ukládají pro každý výskyt portletu zvlášť. Tyto hodnoty se většinou mění v *„edit módu“* portletu a definují se v souboru `portlet.xml`. O samotnou navigaci do editačního módu portletu se stará portál.

```
<portlet>
  ...
  <init-param>
    <name>javax.portlet.faces.defaultViewId.view</name>
    <value>/views/view.xhtml</value>
  </init-param>
```

```
    <init-param>
      <name>javax.portlet.faces.defaultViewId.edit</name>
      <value>/views/preferences.xhtml</value>
    </init-param>
    <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>view</portlet-mode>
      <portlet-mode>edit</portlet-mode>
    </supports>
    ...
    <portlet-preferences>
      <preference>
        <name>memberCode</name>
        <value></value>
      </preference>
      <preference>
        <name>userMode</name>
        <value>MEMBER</value>
      </preference>
    </portlet-preferences>
  <portlet>
```

Výpis 2: Definice edit módu a portlet preferences v portlet.xml

Mód trenér nebo člen je definován portlet preferencí `userMode`. Jak jsem již psal, na propojení s KLUBem se používá pi-kód. Tento kód se standardně načítá od přihlášeného uživatele, ale z důvodu jednoduššího testování jsem se rozhodl přidat do tohoto portletu i možnost určit kód ručně v nastavení portletu (hodnota `memberCode`).

Po deklaraci těchto nastavení v `portlet.xml` jsem dále udělal samotný .xhtml soubor s jednoduchým formulářem a Managed Beanu, která bude tyto nastavení ukládat. Hodnoty těchto preferencí jsou uloženy v EL proměnné `mutablePortletPreferencesValues` a v samotné Managed Beaně se k ní jde dostat pouze skrze EL. To se dá pomocí EL resolveru a nebo jde využít vložení hodnoty do instační proměnné pomocí anotace `@ManagedProperty`.

4.1.4 Určení nasazené verze

Mohlo se stát, že mi tester nahlásil chybu, která už byla opravená, jenže jsem ji ještě nenasadil na testovací server. Proto jsem přidal na viditelné místo v nastavení portletu datum sestavení, aby tester mohl vidět a porovnat ho s datem vyřešení úkolu v bug tracking systému Redmine. Jelikož na sestavování projektu byl používán Maven, využil jsem jeho funkce *filtrování*. V `pom.xml` jsem

nastavil složku, která budou cílem filtrování a tuto složku Maven při sestavení postupně projde a nahradí nalezené výrazy hodnotou.

```
<project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
      apache.org/maven-v4_0_0.xsd">
  ...
  <properties>
    <timestamp>${maven.build.timestamp}</timestamp>
    <maven.build.timestamp.format>dd-MM-yyyy HH:mm</maven.build.
        timestamp.format>
  </properties>
   <build>
      <resource>
        <directory>src/main/resources</directory>
        <filtering>true</filtering>
      </resource>
    </build>
    ...
 </project>
```

Výpis 3: Ukázka nastavení Mavenu s funkcí filtrování a obchůzkou pro datum sestavení

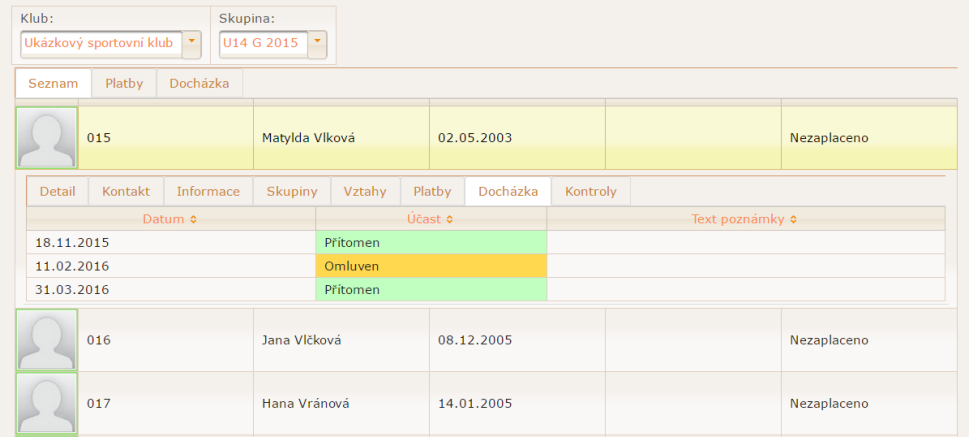
Poté jsem vytvořil soubor a napsal do něj výraz pro datum sestavení. Normálně by výraz vypadal `${maven.build.timestamp}`, jenže kvůli bugu[12] se tato hodnota nefiltruje a musí se musí využít *properties* hodnot. Nakonec mi stačilo obsah tohoto souboru přečíst a vypsat ho v editačním módu portletu.

4.1.5 Rozšíření responzivnosti sloupců

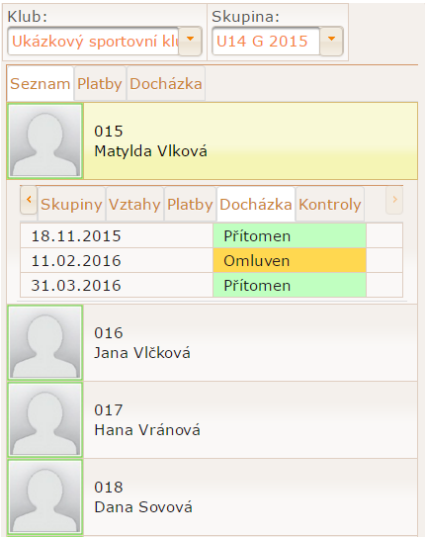
Předpokládalo se, že pohodlne.info bude využíváno na mobilních zařízeních a proto byla nutnost tento požadavek vyřešit. Komponenta `<p:dataTable>` umožňuje automaticky zneviditelnit sloupce podle jejich priority a velikosti výstupní obrazovky. Pouze tato možnost nebyla dostačující, bylo totiž potřeba skrytý obsah zobrazit ve viditelném sloupci. Proto jsem vyvinul rozšíření této funkčnosti.

Toto rozšíření využívá klientské části od `<p:dataTable>` komponenty a oblíbenou JavaScript knihovnu *jQuery*, kterou jsem nemusel ani přidávat, protože tato knihovna je využívána skrze celé PrimeFaces. Při načtení tuto klientskou část rozšiřuji vlastními funkcemi, které požadovanou funkcionalitu splňují.

[12] https://issues.apache.org/jira/browse/MRESOURCES-99

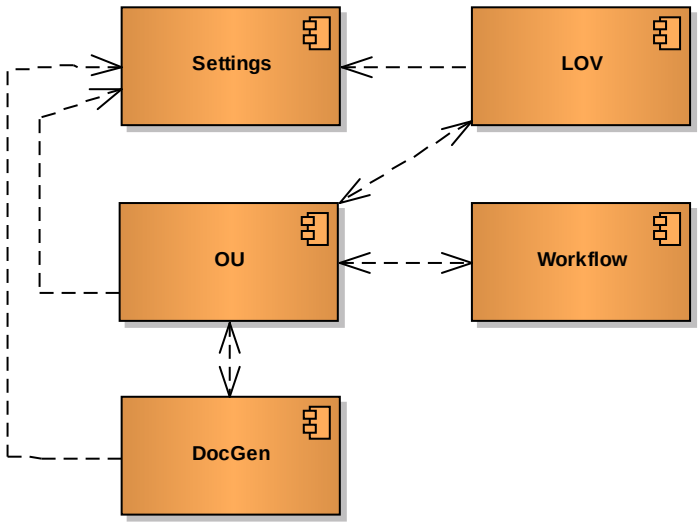


Obrázek 1: Ukázka rozšíření rezponsivnosti sloupců – dekstop verze



Obrázek 2: Ukázka seznamu členů s rozšířením rezponsivnosti tabulky – mobilní verze

Cílem bylo vymyslet, aby se obsah skrytého sloupce naplnil do jiného. Vymyšlené řešení přesně tak funguje a do sloupce označeného třídou `colflowReceiver` se naplní obsah skrytých sloupců se třídou `colReflow`.



Obrázek 3: Schéma modulů v „core“

4.2 Fáze druhá: přesunutí hlavní a nové funkčnosti na portál pohodlne.info

Funkčnost pohodlne.info se velmi podobá jinému projektu a proto vývoj začal analýzou mými kolegy. Byla vymyšlena část „core“, která bude součástí obou projektů, schéma můžete vidět na obrázku č. 3. Samotný „core“ je rozdělen na tyto moduly:

- **DocGen** – generování emailů a dalších dokumentů,
- **LOV** – List of values – defininice a práci s číselníky,
- **OU** – Organizations and Users – rozšíření portálové struktury pro práci s organizacemi a uživateli,
- **Settings** – ukládají se zde různá nastavení pro ostatní části a pro celé moduly i organizace,
- **Workflow** – definice procesů kupř. pro registraci uživatelů.

4.2.1 Modul LOV

Dostal jsem za úkol implementovat modul LOV tzv. seznam hodnot. Modul bude poskytovat tyto seznamy ostatním modulům skrze službu.

Pro tvorbu této služby jsem použil již zmíněný Service Builder. Prvně jsem napsal obsah souboru `service.xml`, spustil nástroj, napsal potřebné implementace metod a znovu spustil nástroj pro propagaci těchto implementovaných metod do příslušných rozhraní.

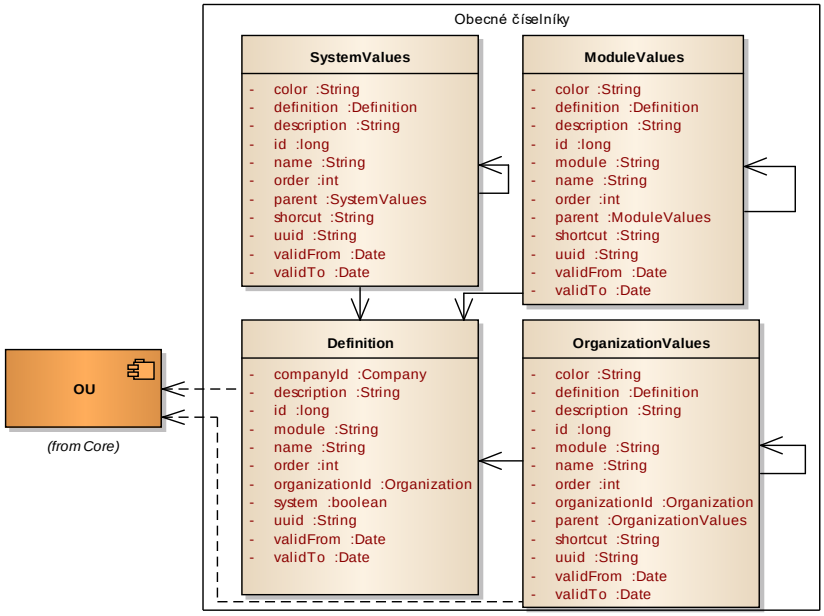
```
   ...
<entity name="Definition" local-service="true" remote-service="false">
   <column name="id" type="long" primary="true" />
   <column name="module" type="String" />
   <column name="uuid" type="String" />
   <column name="name" type="String" />
   <column name="validTo" type="Date" />
   <column name="system" type="boolean" />
   <order by="asc">
     <order-column name="order" />
   </order>
   <finder name="Company" return-type="Collection">
     <finder-column name="companyId" />
   </finder>
   <finder name="U_C" return-type="Definition" unique="true">
     <finder-column name="uuid" case-sensitive="false" />
     <finder-column name="companyId" />
   </finder>
   ...
</entity>
   ...
```

Výpis 4: Ukázka definice služby pro Service builder srkze service.xml

Definici jednoznačně identifikuje kombinace UUID a companyId[13]. Atribut UUID si určuje tvůrce definice.

Samotné hodnoty jsou rozděleny na systémové, modulové a organizační. Výběr ze seznamu probíhá shora dolů. Systémové jsou přepisovány modulovými a modulové jsou přepisovány organizačními. Díky tohoto způsobu výběru si může organizace či modul přepsat systémovou hodnotu nebo ji označit jako neplatnou.

[13]companyId – portál umístěný na server může být dělen na od sebe logicky oddělené instance, každá má svoje companyId



Obrázek 4: Datový model modulu LOV

Tento styl výběru samozřejmě odpovídal implementaci. Prvně odstraním všechny systémové nevalidní položky, uložím je do hašovací tabulky, kde UUID je klíčem. Poté proces probíhá pro modulové a organizační hodnoty stejně.

- Hodnoty, jejichž UUID v hašovací tabulce chybí, se do tabulky přidají.
- Hodnoty, které jsou neplatné a v hašovací tabulce existují, se z tabulky odeberou.
- Hodnoty, které již v tabulce existují a jsou platné, se přepíšou novou hodnotou.

Samotné hodnoty také mohou tvořit stromy. Pro vybrání ze stromů řeším pouze hodnoty bez rodiče. Na získání podhodnot jsem využil návrhového vzoru *Lazy Load*[14], kde hodnota (v tomto případě rodič) sice podhodnoty v sobě uložené nemá, ale ví, kde je získat (skrze službu).

4.2.2 Nová komunikace s KLUBem

Se získanými znalostmi z předmětu Java technologie jsem chtěl využít stejného fyzického umístění aplikací na stejném serveru. Komunikace v Java webových aplikacích probíhá pomocí třídy `RequestDispatcher`[15], stačilo mi objekt třídy implementující toto rozhraní získat a použít.

Při prvních pokusech jsem zjistil, že Apache Tomcat má bezpečností omezení pro komunikace mezi dvěma webovými aplikacemi. Musel jsem tuto komunikaci povolit skrze atribut

[14] http://martinfowler.com/eaaCatalog/lazyLoad.html
[15] https://docs.oracle.com/javaee/7/api/javax/servlet/RequestDispatcher.html

`crossContext` v souboru `context.xml` buď v nastavení Tomcatu a nebo zvlášť pro každou aplikaci.

V takovéto komunikaci se hojně využívají tzv. wrappery, které zapouzdřují původní objekt a navenek se chovají stejně jako on. Jednotlivé metody jdou přepsat a využít jinak. Takto jsem si vytvořil vlastní wrappery. Jeden pro požadavek, který bude mít jakékoliv parametry a druhý pro odpověď, který bude zapisovat data odpovědi do dočasné proměnné, jejíž obsah si poté můžu přečíst, a zároveň neovlivní původní odpověď.

Poté jsem wrappery využil, zjistil jak se v portletu dá `RequestDispatcher` získat a nakonec všechno obalil rozhraním, aby se dalo jednoduše využít i v dalších případech.

4.2.3 Samostatná část pro pohodlne.info

V této části jsem vyvíjel portlety, které budou pouze na portále pohodlne.info. Ostatní projekty již byly založeny, tenhle jsem si založil sám. Projekt jsem nazval „frontend“. Liferay IDE nabízí průvodce založením projektu, kde jsem nastavil název, sestavovací nástroj a typ pluginu.

Jakmile Liferay IDE vygeneroval tento projekt, sjednotil jsem verze knihoven v `pom.xml` s ostatními projekty. Pro založení samotného portletu je třeba ho zaregistrovat v `portlet.xml`, `liferay-portlet.xml` a pokud ho chceme přidávat v portále, musíme ho přidat do `liferay-display.xml`. Plugin typu portlet také obsahuje hook, konfigurovatelný v `liferay-hook.xml`, který jsem použil pro lokalizaci a na úpravu portálového nastavení. Všechny tyto soubory jsou příslušně upravil.

Tentokrát jsem měl k dispozici detailnější návrh GUI. Společně s kolegou jsme analyzovali tento návrh a rozdělili jsme jej na konkrétní portlety s ohledem na co nejvyšší znovupoužitelnost.

Udělal jsem portlet tarifomat, jehož Facelet a Managed Beanu můžete vidět v příloze A a B. Další portlet byl registrace. Tento portlet neslouží jenom k registraci uživatele, ale také bude sloužit k zaregistrování instituce, pro vyzkoušení celého systému. Stejně jako v sekci 4.1.3 jsem udělal tuto registraci skrze nastavení portletu. Dále jsem následoval požadavky, wireframe a podle toho iterativně vyvíjel tento portlet.

5 Využité dovednosti získané v průběhu studia

Tuto odbornou praxi bych nebyl schopen dělat bez pevného základu. Ten obsahuje základní jazykové konstrukce, které se sice skrze různé programovací jazyky trochu liší, ale jejich funkce zůstává stejná (předměty *Programování I* a *II* , *Programovací jazyky I* i *II*, *Tvorba aplikací pro mobilní zařízení I* a *II*). S tímto základem jsou spjaté standardní algoritmy a datové struktury naučené hlavně v předmětech *Algoritmy I* a *II* a jelikož vztah programování k algoritmům je skutečně blízký tak i v ostatních předmětech související s programováním.

Dále je nutnost zmínit znalost některých programovacích paradigmat. V mém případě to bylo hlavně objektově orientované (*Programování II*, *Programovací jazyky I* i *II*, *Vývoj informačních systémů* a *Základy počítačové grafiky*).

Jelikož cílová technologie byla Java, tak byly klíčové předměty *Programovací jazyky I*, pro práci se standardní edicí Javy a *Java technologie*, který mi představil velkou část použitých technologií. Díky podobnosti jazyka C# a Java mi také napomohly znalosti z *Programovací jazyky II*.

Protože Java API[16] a všechny knihovny použité při vývoji jsou založené na návrhových vzorech, předměty *Úvod do softwarového inženýrství* a *Vývoj informačních systémů* i *Základy počítačové grafiky* mě mnohé naučili.

V druhé fázi jsem pomocí nástroje Service Builder pracoval s databází. Ačkoliv tento nástroj většinu práce odstiňuje, koncepty platí stejně (předměty *Úvod do databázových systémů*, *Technologie databázových systémů* a *Databázové a informační systémy*).

5.1 Chybějící znalosti

Osobně bych řekl, že většinu znalostí, které jsem získal během studia se zdají být jako dostačující základ a myslím si, že tohle je přesně, co by studium mělo poskytnout – přehled, povědomí a schopnost se adaptovat. Hodila by se mi větší znalost s verzovacím systémem, jako je kupř. Git, se kterým jsem potřeboval pracovat.

[16] https://docs.oracle.com/javase/8/docs/api/

6 Závěr

Tuto práci bych celou zhodnotil velmi kladně. Moje stávající znalosti jsem si upevnil a aplikoval je v praxi. Navíc jsem získal řadu nových znalostí a hlavně zkušeností.

Zjistil jsem, jak fungují procesy prováděné při vytváření komerčního software. Dále jsem si zkusil, jaké to je následovat již zavedené konvence, jak přistupovat k případným změnám a jak důležité tyto konvence jsou, i když nemusí být optimálně nastaveny.

Samotný Liferay se mi zprvu nelíbil, protože mi přišlo, že naše zvolené technologie nejsou primárně podporovány. Postupem času jsem přišel na to, že ačkoliv nejsou primární, jsou podporované dobře. Navíc velká uživatelská základna a podpora komunity mi umožnila rychle najít řešení na konkrétní problémy. Otevřenost použitých knihoven mi byla dost nápomocná, protože jsem je mohl studovat a lépe jim porozumět.

Martin Vaněk

Literatura

[1] VRK plus s.r.o. - Zakázkový vývoj software a webových aplikací [online]. [cit. 2016-04-08]. Dostupné z: http://www.vrk.cz/

[2] JavaServer Faces Technology Overview. Oracle | Integrated Cloud Applications and Platform Services [online]. [cit. 2016-04-09]. Dostupné z: http://www.oracle.com/technetwork/java/javaee/overview-140548.html

[3] Java Platform, Enterprise Edition: The Java EE Tutorial [online]. 2014 [cit. 2016-04-09]. Dostupné z: https://docs.oracle.com/javaee/7/tutorial/

[4] Liferay Developer Network - Liferay Developer Network [online]. 2014 [cit. 2016-04-09]. Dostupné z: https://dev.liferay.com

A Ukázka ManagedBeany portletu tarifomat

```
@ManagedBean(name = "subscriptionView")
@ViewScoped
public class SubscriptionsViewBean implements Serializable {

    private static final transient Log LOG = LogFactoryUtil.getLog(SubscriptionsViewBean.class);
    private static final long serialVersionUID = 1L;

    private static final String PARAM_MEMBER_COUNT = "memberCount";
    private static final String PARAM_SELECTED_SUBSCRIPTION = "selected-subscription";

    private static long getLayoutPlidViaSettings(LiferayFacesContext ctx, String settingKey) {
        try {
            return LayoutLocalServiceUtil.getFriendlyURLLayout(ctx.getHostGroupId(), false,
                    (String) SettingsUtilsLocalServiceUtil.getValue(settingKey)).getPlid();
        } catch (SystemException | PortalException e) {
            LOG.error(e);
            throw new RuntimeException(e);
        }
    }

    private List<Subscription> subscriptions;

    private String selectedSubscription;
    private int membersCount;

    private long registrationPagePlid;
    private long tryoutPagePlid;

    @PostConstruct
    protected void init() {
        final LiferayFacesContext ctx = LiferayFacesContext.getInstance();
        try {
            this.subscriptions = Subscription.getSubscriptions(ctx.getLocale());

            this.selectedSubscription = Optional.ofNullable(ctx.getRequestParameter(
                    PARAM_SELECTED_SUBSCRIPTION))
                        .orElse(this.subscriptions.get(0).getKey());
            this.membersCount = Optional.ofNullable(ctx.getRequestParameter(PARAM_MEMBER_COUNT)).map(
                    Integer::parseInt)
                        .orElse(1);
        } catch (SystemException e) {
            LOG.error(e);
            throw new RuntimeException(e);
        }
    }

    private LiferayPortletURL createRegistrationUrl(Subscription subscription, LiferayFacesContext ctx, long plid) {
        LiferayPortletURL registrationURL;
        registrationURL = PortletURLFactoryUtil.create(ctx.getPortletRequest(), PIConstants.
                REGISTRATION_PORTLET_ID,
                    plid, PortletRequest.RENDER_PHASE);
        registrationURL.setParameter(PARAM_SELECTED_SUBSCRIPTION, subscription.getKey());
        registrationURL.setParameter(PARAM_MEMBER_COUNT, String.valueOf(this.membersCount));
```

```
        return registrationURL;
    }

    public int getMembersCount() {
        return this.membersCount;
    }

    public Integer getPrice(Subscription subscription) {
        return subscription.getPrices().get(this.membersCount);
    }

    public String getRegistrationUrl(Subscription subscription) {

        LiferayFacesContext ctx = LiferayFacesContext.getInstance();
        if (this.registrationPagePlid == 0L) {
            this.registrationPagePlid = getLayoutPlidViaSettings(ctx, SettingKeyConstants.
                REGISTRATION_FRIENDLY_URL);
        }
        LiferayPortletURL registrationURL = createRegistrationUrl(subscription, ctx, this.registrationPagePlid);
        return registrationURL.toString();
    }

    public String getSelectedSubscription() {
        return this.selectedSubscription;
    }

    public List<Subscription> getSubscriptions() {
        return this.subscriptions;
    }

    public String getTryoutUrl(Subscription subscription) {
        LiferayFacesContext ctx = LiferayFacesContext.getInstance();
        if (this.tryoutPagePlid == 0L) {
            this.tryoutPagePlid = getLayoutPlidViaSettings(ctx, SettingKeyConstants.TRYOUT_FRIENDLY_URL);
        }
        LiferayPortletURL registrationURL = createRegistrationUrl(subscription, ctx, this.tryoutPagePlid);
        return registrationURL.toString();
    }

    public void onMemberSliderChange(SlideEndEvent event) {
        this.setMembersCount(event.getValue());
    }

    public void setMembersCount(int membersCount) {
        this.membersCount = membersCount;
    }

    public void setSelectedSubscription(String selectedSubscription) {
        this.selectedSubscription = selectedSubscription;
    }

}
```

Výpis 5: SubscriptionsViewBean.java

B Ukázka Faceletu portletu tarifomat

```
<ui:composition
    xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:p="http://primefaces.org/ui"
    xmlns:jsf="http://xmlns.jcp.org/jsf">
    <h:outputText
        id="membersCountOutput"
        value="#{subscriptionView.membersCount}" />
    <h:inputHidden
        id="membersCountInput"
        value="#{subscriptionView.membersCount}" />
    <p:slider
        display="membersCountOutput"
        for="membersCountInput"
        minValue="1"
        maxValue="1000">
        <p:ajax
            update="@next"
            partialSubmit="true"
            listener ="#{subscriptionView.onMemberSliderChange}" />
    </p:slider>
    <p:outputPanel styleClass="clearfix">
        <ui:repeat
            var="sub"
            value="#{subscriptionView.subscriptions}">
            <p:panelGrid
                columns="1"
                layout="grid"
                style="float:  left ;">
                <f:facet  name="header">
                    <p:commandLink
                        rendered="#{!showSubscriptionLinks}"
                        action="#{subscriptionView.setSelectedSubscription(sub.key)}"
                        partialSubmit="true"
                        process="@this"
                        update="@parent:@parent:@parent:@parent">
                        <h:outputText
                            value="#{sub.name}"
                            style="#{sub.key eq subscriptionView.selectedSubscription?'color: red ;' : ''} " />
                    </p:commandLink>
                    <h:outputText
                        rendered="#{showSubscriptionLinks}"
                        value="#{sub.name}" />
                </f:facet>
                <h:outputText value="#{subscriptionView.getPrice(sub)} " />
                <ul class="pi-subscriptions-advantages">
                    <ui:repeat
                        var="advantage"
                        value="#{sub.advantages}">
                        <li>#{advantage}</li>
                    </ui:repeat>
```

```
                </ul>
                <p:link
                    rendered="#{showSubscriptionLinks}"
                    href="#{subscriptionView.getRegistrationUrl(sub)}"
                    value="#{i18n['buy']}">
                </p:link>
                <p:link
                    rendered="#{showSubscriptionLinks}"
                    href="#{subscriptionView.getTryoutUrl(sub)}"
                    value="#{i18n['tryout']}">
                </p:link>
            </p:panelGrid>
        </ui:repeat>
    </p:outputPanel>
</ui:composition>
```

Výpis 6: subscriptions.xhtml